World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:8, No:2, 2014

Frequent and Systematic Timing Enhancement of Congestion Window in Typical Transmission Control Protocol

Ghassan A. Abed, Akbal O. Salman, Bayan M. Sabbar

***Abstract*—**Transmission Control Protocol (TCP) among the wired and wireless networks, it still has a practical problem; where the congestion control mechanism does not permit the data stream to get complete bandwidth over the existing network links. To solve this problem, many TCP protocols have been introduced with high speed performance. Therefore, an enhanced congestion window (*cwnd*) for the congestion control mechanism is proposed in this article to improve the performance of TCP by increasing the number of cycles of the new window to improve the transmitted packet number. The proposed algorithm used a new mechanism based on the available bandwidth of the connection to detect the capacity of network path in order to improve the regular clocking of congestion avoidance mechanism. The work in this paper based on using Network Simulator 2 (NS-2) to simulate the proposed algorithm.

***Keywords*—**TCP, cwnd, Congestion Control, NS-2.

I. Introduction

THE network simulator NS-2 [1] is a freely accessible object-oriented and discrete–event network simulator, which provides a structure for constructing the network prototype. NS-2 identifies data as input parameters, analyzing data output and giving outcomes and results. Two main reasons for the wide impact of NS-2 are as follows, the first is because it is free, where that fits researchers in laboratories and universities, and the second reason is the huge range of network modules and objects that can implemented by NS-2 [2].

The TCP protocol is the most widely used protocol for wired and wireless systems, although TCP was not originally designed for real time applications and not for wireless networks. Then we need to develop new TCP versions, or at least choose a suitable TCP variant for each new network to be more efficient and more reliable with this network. For the first time to the congestion control mechanisms in early 1980 and was designed primarily to stop the collapse with traffic congestion typical of that era. In recent years, increased volume of traffic generated by real-time applications, a high proportion of the large transport package and congestion control in more often than not opt for these types of applications [3].

Ghassan A. Abed is with Ministry of Science and Technology, Baghdad, Iraq (e-mail: dr.ghassan.abed@ieee.org).
Akbal O. Salman is with Technical College, Al-Musaib, Iraq (e-mail: it.akbal@yahoo.com).
Bayan M. Sabbar is with Al-Nahrain University, Baghdad, Iraq (e-mail: dr_b2012@yahoo.co.uk).

Besides, the systems can fit into the new system is uncontrolled, so much the overall productivity of the oscillating flow and one which, in turn, can result in poor application performance. Apart from concerns about the network level, and those types of applications a lot of care, end-to-end delay in the speed and smooth congestion control, this does not fit the traditional schemes.

In this research, we will investigate improving the performance of the new congestion control algorithm, developed to be able to transfer high rate of packet over large bandwidth low-latency platform. When using TCP over the cellular infrastructure, and the result is that both times of end-to-end production and use of radio links is very weak. This is because the dynamic characteristics of TCP and wireless connections do not fit well together [4].

TCP limits transmission rate by controlling the send congestion window size, which is the number of packets that can be transmitted in the flow. Generally, the time between the submission and receipt of ACK packet is Round-Trip Time (RTT). The TCP sender can send up to the congestion window size of data packets during one RTT. Once send the TCP window size of data packets, it can send the new data packet only after it is reached to some of the ACKs to the sender. Therefore, the average rate of more than TCP RTT is almost the size of the window divided by the RTT [5].

The most important concept in TCP congestion control is the congestion window, where the window is the amount of data that have been sent, but which have not yet received any confession. Congestion window constant means that the broadcast packet and a new one for each ACK that is received while the control is in the rate of transmission indirectly by adjusting the size of the congestion. It has been documented standard way to do so in RFC2581 [6], usually referred to as TCP Reno. The other common TCP versions are Newneno [7] and TCP with selective ACK (Sack) [8].

In order to improve TCP performance, many TCP variants, which mainly differ with each other on the functions of congestion control, have been proposed. Currently, there are six TCP source variants; Tahoe, Reno, Newreno, Fack, Sack, and Vegas, adding many other extended variants. Each variant has been a private congestion control algorithm or developed a previous algorithm, to be efficient and reliable with a new end-to-end application.

II. Proposed Mechanism

Primary role, to control congestion, adjusts the window of

data transmission at sender side in such a way that is preventing buffer overflow in the recipient, but also in the intermediate routers. To achieve this, TCP used another variable to control the congestion window. Congestion control represents a number of segments of appreciation that can be injected into the network without causing congestion. The challenge is to take advantage of the available space in the store network routers. Routers do not participate in the TCP layer and the chip cannot be used to adjust the TCP ACK frame. To resolve this problem, TCP assumes network congestion as the retransmission timer expires, and that it interacts with the network congestion by adjusting the congestion window using two algorithms, slow start and congestion avoidance, as shown in Fig. 1.

In the slow start phase, and when the connection is established, is first set the value of *cwnd* to one and then each received ACK value is updated to: *cwnd*= *cwnd*+1 which means doubling the *cwnd* per RTT.

The rapid growth of *cwnd* continues until the packet loss was observed, causing the value of slow-start threshold (*ssthresh*) is updated to: *ssthresh*=*cwnd*/2. After losing the packet, the connection starts from slow start again by set *cwnd*=1, and is increasing exponentially until the window is equal to *ssthresh*, the estimate of available bandwidth in the network.

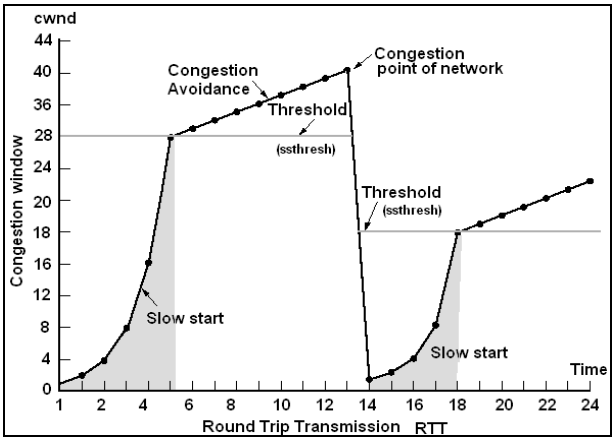

Fig. 1 TCP slow-start and congestion avoidance

At this point, in goes to the congestion avoidance phase, where the value of *cwnd* is less aggressive with the pattern: *cwnd*=*cwnd*+1/*cwnd*, which implies a linear rather than exponential growth. And will continue to increase until the written disclosure of packet loss. The new mechanism proposed in this article introduces new congestion avoidance algorithms by estimate the predictable throughput with the prospect of higher productivity, regardless of the level of congestion.

The new algorithm depends on using the available capacity on the network links to detect the increasing or decreasing the size of the congestion window to obtain an adaptive congestion avoidance mechanism.

The evaluation and representation of the new algorithm performed using NS-2 to analyze the performance of the proposed mechanism over many experiments.

Certainly, the proposed algorithm provided an increment in the network path about 20-30%, and that allows growing the bottleneck capacity too, even the network suffers from congestion. In proposed mechanism, we used classic exponential increment to in slow-start phase. Where congestion window *cwnd* less than slow-start threshold *ssthresh* and the window size increases by one, as explained in (1):

if (*cwnd* < *ssthresh*)
Then
$$cwnd=cwnd+1 \quad (1)$$

TCP sender update's congestion window size *cwnd* in the congestion avoidance phase is according to the following equation when it receives an ACK packet from the receiver TCP as shown in (2):

$$cwnd=cwnd+(f/cwnd) \quad (2)$$

where *f* is control parameter. From this equation, we expect that the congestion window size increases by *f* segments in every RTT. The main function of the proposed mechanism is to regulate *f* dynamically and adaptively, while the original TCP Reno uses a fixed value of *f*=1. In the rest of this subsection, we explain how to change *f* to the network condition and the current throughput of the TCP connection. The classic algorithm's code is as follows:

cwnd_ = *cwnd_* + (*f* / *cwnd_*)
Where:
*cwnd_* : is the real *cwnd* variable in NS-2.
*ssthresh_*: is the real *ssthresh* variable in NS-2.
f : control parameter, and f=1 in Reno.

In proposed mechanism, we need to update *f* for (2) when the sender of TCP receives a new ACK. By this ACK-based mechanism, the proposed mechanism can accommodate the fluctuating of RTTs of the network path. In fact we depend on four main parameters to updates *f* values every RTT. These parameters illustrated below:

- *ssthresh*: slow-start threshold of network path.
- *cwnd*: the last value of *cwnd*.
- wnd_const: packets per RTT.
- K_parameter: k parameter in binomial controls.

By using these parameters for determining *f*, the degree of increase of congestion window size becomes too large when the current throughput of a TCP connection is far below the target throughput. The large values of *f* will cause bursty packet losses in the network and resulting in performance degradation due to retransmission timeouts. On the other hand, when the network has sufficient residual bandwidth, the degree of increase of the congestion window size becomes smaller than one.

Therefore, we limit the maximum and minimum values of *f*. For that, every RTT, the new value of *f* becomes:

$$f=ssthresh*\text{wnd\_const} \quad (3)$$

In (3), *f* will equal the available bandwidth; this will adjust the sending interval of the data packets according to the available bandwidth of the network path. Next, when the proposed formula in (3) has obtained a larger throughput as explained, we need to minimize the value off to avoiding packet losses. The control of this problem by divides the last value off obtained from (3) by the previous *cwnd* value multiplied by the exponent *cwnd* to the *k* parameter in binomial control as shown in (4):

$$f=f/(cwnd*\text{pow}(cwnd,\text{k\_parameter})) \quad (4)$$

This means, the proposed mechanism steals bandwidth from competing flows in the network in order to achieve the bandwidth required by the upper-layer application. In summary, the proposed mechanism updates *f* using the following formula when the TCP sender receives a new ACK:

```
f=wnd_const_*ssthresh_/cwnd_
        *pow(cwnd_, k_parameter_);

cwnd_ = cwnd_ + ( f / cwnd_);
```

III. Results and Discussion

Firstly, we check whether the new mechanism can be effective to the bandwidth available in the network path. In these experiments, we performed the data transfer using the new TCP by change the bottleneck bandwidth of the network and keep all other parameters. As shown in Fig. 2, we used a bandwidth of 100 Mbps and test the behavior of *cwnd* of new TCP and compared it with TCP Reno. The measurement results of *cwnd* during the experiment explain that the congestion window of new TCP performed well compared with Reno congestion window, so the clocking of new TCP is also better than Reno clocking; in addition we have a smooth slow-start phase like Reno.

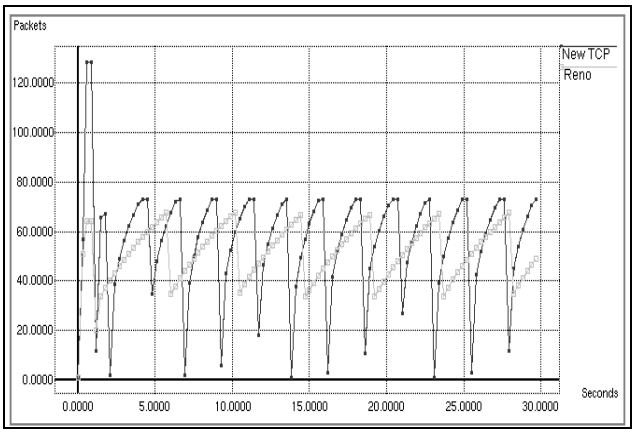

Fig. 2 Behavior of new mechanism with Reno (BW=100Mbps)

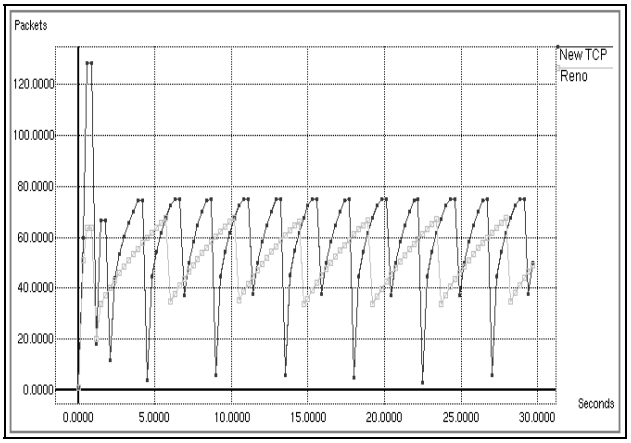

Fig. 3 Behavior of new mechanism with Reno (BW=200 Mbps)

In Fig. 3, we can note the large increment in *sound* between new TCP and Reno, and this difference because that we increased the bandwidth from 100Mbps in firs experiment (Fig. 2) to 200 Mbps. This increment permits to the new mechanism to open *cwnd* reaches to 75 packets while Reno kept the same packet level in two experiments of less than 65 packets. The reason beyond to the difference in the mechanism used in Reno and new TCP, as we explained that because Reno used *f*=1, while new TCP estimates the available capacity of the network path.

The performance variation of two mechanisms appears clearly in the next test, when we used a bandwidth of 500 Mbps as shown in Fig. 4. This leads to getting a *cwnd* exceeded 80 packets provided by new TCP, with constant window size in Reno. Furthermore, Fig. 4 shows that the available bandwidth will gives a throughput of cross traffic. That is, TCP Reno cannot be used for background data transfer. That means that the new TCP will increase the *cwnd* in same time that it will not decrease the throughput of the co-existing foreground traffic.

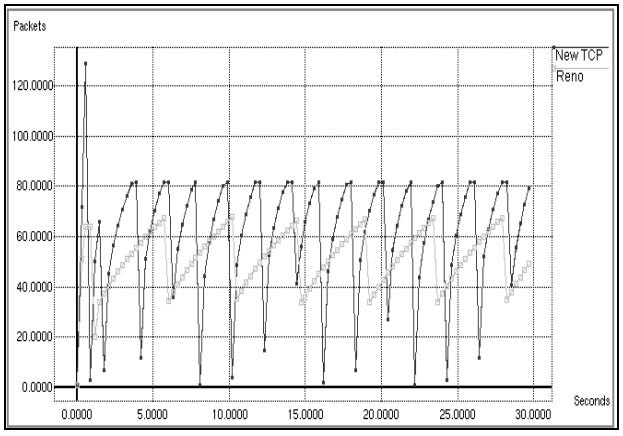

Fig. 4 Behavior of new mechanism and Reno (BW=500 Mbps)

Furthermore, Figs. 2-4, show that the *cwnd* of new TCP is closest to the available bandwidth. Therefore, the new mechanism can utilize the available bandwidth better than Reno.

These results clearly show the proposed mechanism which utilizes the available bandwidth information obtained by the new formula, performs well in the experimental network environment. For this reason and from previous experiments, it is not efficient to use Reno or any other classic TCP variants over high bandwidth networks because of the limited performance of congestion control for these variants and that is the main reason to develop new mechanisms has more efficiency.

IV. Conclusion

This article proposed a new mechanism to improve the clocking of the congestion window in TCP congestion Control. The new mechanism modifies the degree of increase of the congestion window size of a TCP connection in the congestion avoidance phase by using the information of the available bandwidth of the network path. The enhanced mechanism provides about 125 packets as a maximum congestion point, but Reno kept the same previous value of 65 packets. On the other hand, the new TCP gave a performance speed more than twice that obtained from Reno, when it completes one clocking cycle of *cwnd* in Reno, we obtained more than two cycles in new TCP. That means we will send more than double amount of packets in new mechanism.

V. Future Work

The future work of this article will emphasize to modify the slow-start mechanism of the TCP to improve the congestion control of standard TCP variants.

References

[1] Simulator, N., ns-2. 1989.
[2] J. Olsén, Stochastic modeling and simulation of the TCP protocol. 2003: Matematiska Institutionen.
[3] G. A. Abed, M. Ismail, and K. Jumari, "Distinguishing Employment of Stream Control Transmission Protocol over LTE-Advanced Networks," *Research Journal of Information Technology*, vol. 3, pp. 207-214, 2011.
[1] N. Moller. Automatic Control in TCP over Wireless. Licentiate Thesis, Stockholm, Sweden, 2005.
[2] S. Choi. Design and Analysis for TCP-Friendly Window-based Congestion Control. In: PhD Thesis, *University College London*, 2006.
[3] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. RFC 2581, April 1999.
[4] S. Floyd and T. Henderson. The NewReno modification to TCP's fast recovery algorithm. RFC 2582, April 1999.
[5] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgment options. RFC 2018, October 1996.
[6] S. Floyd, J. Mahdavi, M. Mathis, and M Podolsky. An extension to the selective acknowledgement (SACK) option for TCP. RFC 2883, July 2000.
[7] G. A. Abed, M. Ismail, and K. Jumari, "Appraisal of Long Term Evolution System with Diversified TCP's," *Modelling Symposium (AMS), 2011 Fifth Asia*, 2011, pp. 236-239.
[8] G. A. Abed, M. Ismail, and K. Jumari, "Traffic Modeling of LTE Mobile Broadband Network Based on NS-2 Simulator," *Computational Intelligence, Communication Systems and Networks (CICSyN), 2011 Third International Conference on*, 2011, pp. 120-125.